پیاده سازی الگوریتم Naive Bayes Classifier

در این پروژه ما قصد داریم تا الگوریتم Naive Bayes Classifier را پیاده سازی کنیم.

01224170 :شماره دانشجویی

Naive Bayes Classifier :نام پروژه

۱. کتابخانه ها و توابع اولیه مورد نیاز

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
import os, sys, glob
import numpy as np
import pandas as pd
import matplotlib as mpl
import math

PWD = '.'
if os.path.exists('/content/drive'):
    PWD = '/content/drive/MyDrive'

CATEGORIES = 2

parts_ids = {

}

COUNTER_MAX = 5

ADDRS_MAX = 0
ADDITION_MAX = 2

TRAIN_DATA_FILE_PATH = 'Naive_train.csv'
TEST_DATA_FILE_PATH = 'Naive_test.csv'
TRAIN_DATA_FILE_PATH = 'train-parts.csv'
```

در کد بالا ابتدا پوشه درایو را ست می‌کنیم.

۲. توابع کمکی

```
def delete_numbers(from_string):
    output = 0
    for d in '0123':
        if d in from_string:
            output = 1
    return output

def split(input_csv_path,output_path):
    with open(input_path,'r') as fin:
        for qid in fin:
            if qid.strip() == 'qwertyuiop':
            break
    df = pd.read_csv(input_csv_path,header=None,sep='|')
    df.columns = ['id', 'title', 'description', 'categories']
    df['title'] = df['title'].apply(lambda x: str(x))
    df['description'] = df['description'].apply(lambda x: str(x))
    df['categories'] = df['categories'].apply(lambda x: int(x))
    df = df[(df['title'] != '') & (df['description'] != '')]
    df.to_csv(output_path,sep='|',header=False,index=False)

split(TRAIN_DATA_FILE_PATH,'train-parts.csv')
split(TEST_DATA_FILE_PATH,'test-parts.csv')
```

در کد بالا توابع split و delete_numbers تعریف شده‌اند. تابع split فایل ورودی را به فایل train-parts.csv و test-parts.csv تبدیل می‌کند.

۳. خواندن داده‌ها

```
train_df = pd.read_csv('train-parts.csv',sep='|')
test_df = pd.read_csv('test-parts.csv',sep='|')
```

```
train_df.shape
```

```
(250000, 4)
```

```
train_df.head()
```

	id	title	description	categories
0	0	[illegible]	[illegible]	1
1	1	[illegible]	[illegible]	1
2	2	[illegible]	[illegible]	1
3	3	[illegible]	[illegible]	1
4	4	[illegible]	[illegible]	1

```
train_df.tail()
```

	id	title	description	categories
249995	249995	[illegible]	[illegible]	1
249996	249996	[illegible]	[illegible]	1
249997	249997	[illegible]	[illegible]	1
249998	249998	[illegible]	[illegible]	1
249999	249999	[illegible]	[illegible]	1

250000 rows × 4 columns

```
test_df.shape
```

```
(40000, 4)
```

```
test_df.head()
```

	id	title	description	categories
0	0	[illegible]	[illegible]	1
1	1	[illegible]	[illegible]	1
2	2	[illegible]	[illegible]	1
3	3	[illegible]	[illegible]	1
4	4	[illegible]	[illegible]	1

```
test_df.tail()
```

	id	title	description	categories
39995	39995	[illegible]	[illegible]	1
39996	39996	[illegible]	[illegible]	1
39997	39997	[illegible]	[illegible]	1
39998	39998	[illegible]	[illegible]	1
39999	39999	[illegible]	[illegible]	1

40000 rows × 4 columns

۴. پیش پردازش

```
train_df['text'] = train_df['title'] + train_df['description']
train_df.drop(columns=['title', 'description'], inplace=True)
test_df['text'] = test_df['title'] + test_df['description']
test_df.drop(columns=['title', 'description'], inplace=True)
```

```
train_df.head()
```

	categories	text_ids
0	1	[illegible]
1	1	[illegible]
2	1	[illegible]
3	1	[illegible]
4	1	[illegible]

```
train_df.tail()
```

	categories	text_ids
249995	1	[illegible]
249996	1	[illegible]
249997	1	[illegible]
249998	1	[illegible]
249999	1	[illegible]

250000 rows × 2 columns

```
test_df.head()
```

	categories	text_ids
0	1	[illegible]
1	1	[illegible]
2	1	[illegible]
3	1	[illegible]
4	1	[illegible]

```
test_df.tail()
```

	categories	text_ids
39995	1	[illegible]
39996	1	[illegible]
39997	1	[illegible]
39998	1	[illegible]
39999	1	[illegible]

40000 rows × 2 columns

```
def dictionary_set_handler(element, dic, add_value):
    if element in dic:
        dic[element] += add_value
    else:
        dic[element] = dic[element] + add_value

def initialize(value):
    return value

class Utilities:
    def __init__(self):
        self.category_count_word = {}
        self.feature_set_per_category = {}
        self.category_word_count = {}
        self.category_sample_int = {}
        self.sample_count = 0
        self.total_sample = 0

    def map_function(self, word, category):
        if word in self.feature_set_per_category.keys():
            set = self.feature_set_per_category[category]
            if word in set:
                return 0
            else:
                self.feature_set_per_category[category].add(word)
                return 1
        else:
            self.feature_set_per_category[word] = {}
            self.feature_set_per_category[category].add(word)
            return 1

    def reduce_function(self, data_frame):
        for word, row in data_frame.iterrows():
            for c in range(0, CATEGORIES):
                category = row[c+1]
                if category not in self.category_word_count.keys():
                    self.category_word_count[category] = {}
                    self.category_word_count[category][word] = 0
                else:
                    if word not in self.category_word_count[category].keys():
                        self.category_word_count[category][word] = 0
                    self.category_word_count[category][word] = self.category_word_count[category][word] + row[0]
        return data_frame

    def count(self, data_frame):
        for c in range(0, CATEGORIES):
            self.category_sample_int[c] = 0
            self.category_count_word[c] = 0
            self.feature_set_per_category[c] = set()
        for index, row in data_frame.iterrows():
            self.sample_count = self.sample_count + 1
            category = row[1]
            if category not in self.category_sample_int.keys():
                self.category_sample_int[category] = 1
                self.category_count_word[category] = 0
                self.feature_set_per_category[category] = set()
            else:
                self.category_sample_int[category] = self.category_sample_int[category] + 1
            self.total_sample = self.total_sample + 1
        return data_frame

class Probabilities:
    def __init__(self, alpha):
        self.alpha = alpha
        self.vocabulary = {}
        self.alpha = alpha

    def calculate_prob(self, train_utilities):
        self.vocabulary = len(train_utilities.feature_set_per_category.keys())
        self.calculate_likelihood_prob(train_utilities)
        self.calculate_prior_prob(train_utilities)

    def calculate_prior_prob(self, train_utilities):
        self.prior = {}
        for category in train_utilities.category_sample_int:
            self.prior[category] = train_utilities.category_sample_int[category] / train_utilities.total_sample

    def calculate_likelihood_prob(self, train_utilities):
        self.likelihood = {}
        for category in train_utilities.category_count_word.keys():
            if category not in self.likelihood:
                self.likelihood[category] = {}
            for word in train_utilities.feature_set_per_category[category]:
                category_word_count = train_utilities.category_count_word[category]
                word_count = train_utilities.category_word_count[category][word]
                self.likelihood[category][word] = (word_count + self.alpha) / (category_word_count + self.alpha * self.vocabulary)

    def calculate_prob_probability(self, train_utilities, evidence, category):
        prob = train_utilities.category_sample_int[category] / train_utilities.total_sample
        for word in evidence:
            if word in train_utilities.feature_set_per_category[category]:
                prob = prob * self.likelihood[category][word]
            else:
                prob = prob * self.alpha / (train_utilities.category_count_word[category] + self.alpha * self.vocabulary)
                return prob
        return prob
    def predict(self, train_utilities, evidence):
        return max(train_utilities.category_sample_int, key=lambda x: self.calculate_prob_probability(train_utilities, evidence, x))
    def get_prob(self, train_utilities, evidence, category):
        prob = self.calculate_prob_probability(train_utilities, evidence, category)
        if prob > max_prob and max_prob != 0:
            return 1.0
        elif max_prob == 0:
            return 1.0
        else:
            return prob / max_prob

train_utilities = Utilities()
train_utilities.initialize_utilities(train_df)

train_probabilities = Probabilities(NAIVE_ALPHA)
train_probabilities.calculate_probabilities(train_utilities)

addition_train_probabilities = Probabilities(ADDITION_ALPHA)
addition_train_probabilities.calculate_probabilities(train_utilities)
```

Bigrams

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

fig = plt.figure(figsize=(8, 7), dpi=100, facecolor='w', edgecolor='k')
# x = np.arange(len(CATEGORIES))
# ax = fig.add_subplot(111)
ax1 = subplot(2, 2, 1)
ax1.bar(train_utilities.keys(), train_utilities.values())
ax1.set_ylabel('Freq')
ax1.set_xlabel('Class')
ax1.set_title('Class Frequency')
ax2 = subplot(2, 2, 2)
ax2.bar(train_utilities.keys(), train_utilities.category_sample_int)
ax2.set_ylabel('Freq')
ax2.set_xlabel('Class')
ax2.set_title('Sample Frequency')
ax3 = subplot(2, 2, 3)
ax3.bar(train_utilities.keys(), train_utilities.category_count_word, color='darkred', width = 0.4)
ax3.set_ylabel('Freq')
ax3.set_xlabel('Class')
ax3.set_title('Word Frequency')
ax4 = subplot(2, 2, 4)
ax4.bar(train_utilities.keys(), train_utilities.feature_set_per_category.values())
ax4.set_ylabel('Freq')
ax4.set_xlabel('Class')
ax4.set_title('Feature Set Frequency')
fig.tight_layout()
plt.show()
```


۵. پیاده سازی الگوریتم

```
train_probabilities.get_prob(train_utilities,train_df.iloc[1,2],1)
```

```
1.0
```

```
train_probabilities.get_prob(train_utilities,train_df.iloc[1,2],0)
```

```
1.0
```

```
train_probabilities.get_prob(addition_train_probabilities,train_df.iloc[1,2],1)
```

```
1.0
```

```
train_probabilities.get_prob(addition_train_probabilities,train_df.iloc[1,2],0)
```

```
1.0
```

```
train_df['Normal_Mode'] = train_df.apply(lambda_probabilities.calculate_class, axis = 1)
train_df['Additive_Mode'] = train_df.apply(addition_train_probabilities.calculate_class, axis = 1)
train_df.drop(columns=['text_ids'])
```

	categories	text_ids	Normal_Mode	Additive_Mode
0	1	[illegible]	1.0	1.0
1	1	[illegible]	1.0	1.0
2	1	[illegible]	1.0	1.0
3	1	[illegible]	1.0	1.0
4	1	[illegible]	1.0	1.0
...	...	...	...	...
249995	1	[illegible]	1.0	1.0
249996	1	[illegible]	1.0	1.0
249997	1	[illegible]	1.0	1.0
249998	1	[illegible]	1.0	1.0
249999	1	[illegible]	1.0	1.0

250000 rows × 4 columns

۶. ارزیابی

```
def set_class_scores(c, single_class_result, normal_all_detected_class, additive_all_detected_class):
    global single_class_result, normal_all_detected_class, additive_all_detected_class
    single_class_result['Normal'] = single_class_result['Normal'] + 1
    normal_all_detected_class[c] = normal_all_detected_class[c] + 1
    if single_class_result['Normal'] == 1:
        if normal_precision > normal_recall:
            pass
        else:
            normal_f1 = 2 * (normal_precision * normal_recall) / (normal_precision + normal_recall)
    additive_precision = additive_correct_detected_class / additive_all_detected_class
    additive_recall = additive_correct_detected_class / test_data_categories[c]
    if additive_precision > additive_recall:
        pass
    else:
        single_class_result['Precision'] = 2 (normal_precision * 100)
        single_class_result['Recall'] = 2 (normal_recall * 100)
        single_class_result['F1'] = 2 (normal_f1 * 100)
        single_class_result['Additive_Mode'] = 100 - additive_recall * 100
        single_class_result['Normal_Mode'] = 100 - normal_recall * 100

single_class_result = {'Normal': 0, 'Additive': 0}

normal_accuracy = 0.0
normal_correct = 0.0
for c in range(0, CATEGORIES):
    single_class_result = {'Normal': 0, 'Additive': 0}
    set_class_scores(c, single_class_result, normal_all_detected_class, additive_all_detected_class)
    normal_correct = normal_correct + normal_all_detected_class[c]
    normal_accuracy = normal_correct / test_data_categories[c]
    print('Class ', c, ' Normal Accuracy: ', normal_accuracy, ' %')
```

```
Class  0  Normal Accuracy:  0.0  %
Class  1  Normal Accuracy:  0.0  %
```

```
single_class_result = {'Normal': 0, 'Additive': 0}
normal_accuracy = 0.0
normal_correct = 0.0
for c in range(0, CATEGORIES):
    single_class_result = {'Normal': 0, 'Additive': 0}
    set_class_scores(c, single_class_result, normal_all_detected_class, additive_all_detected_class)
    normal_correct = normal_correct + normal_all_detected_class[c]
    normal_accuracy = normal_correct / test_data_categories[c]
    print('Class ', c, ' Normal Accuracy: ', normal_accuracy, ' %')
```

```
Class  0  Normal Accuracy:  0.0  %
Class  1  Normal Accuracy:  0.0  %
```

```
def set_class_scores(c, single_class_result, normal_all_detected_class, additive_all_detected_class):
    global single_class_result, normal_all_detected_class, additive_all_detected_class
    single_class_result['Normal'] = single_class_result['Normal'] + 1
    normal_all_detected_class[c] = normal_all_detected_class[c] + 1
    if single_class_result['Normal'] == 1:
        if normal_precision > normal_recall:
            pass
        else:
            normal_f1 = 2 * (normal_precision * normal_recall) / (normal_precision + normal_recall)
    additive_precision = additive_correct_detected_class / additive_all_detected_class
    additive_recall = additive_correct_detected_class / test_data_categories[c]
    if additive_precision > additive_recall:
        pass
    else:
        single_class_result['Precision'] = 2 (normal_precision * 100)
        single_class_result['Recall'] = 2 (normal_recall * 100)
        single_class_result['F1'] = 2 (normal_f1 * 100)
        single_class_result['Additive_Mode'] = 100 - additive_recall * 100
        single_class_result['Normal_Mode'] = 100 - normal_recall * 100

single_class_result = {'Normal': 0, 'Additive': 0}

normal_accuracy = 0.0
normal_correct = 0.0
for c in range(0, CATEGORIES):
    single_class_result = {'Normal': 0, 'Additive': 0}
    set_class_scores(c, single_class_result, normal_all_detected_class, additive_all_detected_class)
    normal_correct = normal_correct + normal_all_detected_class[c]
    normal_accuracy = normal_correct / test_data_categories[c]
    print('Class ', c, ' Normal Accuracy: ', normal_accuracy, ' %')
```

```
Class  0  Normal Accuracy:  0.0  %
Class  1  Normal Accuracy:  0.0  %
```

همانطور که مشاهده می‌کنید مقدار Accuracy برای هر دو مدل Additive و Normal برابر با صفر می‌باشد.

```
single_class_result = {'Normal': 0, 'Additive': 0}
normal_accuracy = 0.0
normal_correct = 0.0
for c in range(0, CATEGORIES):
    single_class_result = {'Normal': 0, 'Additive': 0}
    set_class_scores(c, single_class_result, normal_all_detected_class, additive_all_detected_class)
    normal_correct = normal_correct + normal_all_detected_class[c]
    normal_accuracy = normal_correct / test_data_categories[c]
    print('Class ', c, ' Normal Accuracy: ', normal_accuracy, ' %')
```

```
Class  0  Normal Accuracy:  0.0  %
Class  1  Normal Accuracy:  0.0  %
```

```
def set_class_scores(c, single_class_result, normal_all_detected_class, additive_all_detected_class):
    global single_class_result, normal_all_detected_class, additive_all_detected_class
    single_class_result['Normal'] = single_class_result['Normal'] + 1
    normal_all_detected_class[c] = normal_all_detected_class[c] + 1
    if single_class_result['Normal'] == 1:
        if normal_precision > normal_recall:
            pass
        else:
            normal_f1 = 2 * (normal_precision * normal_recall) / (normal_precision + normal_recall)
    additive_precision = additive_correct_detected_class / additive_all_detected_class
    additive_recall = additive_correct_detected_class / test_data_categories[c]
    if additive_precision > additive_recall:
        pass
    else:
        single_class_result['Precision'] = 2 (normal_precision * 100)
        single_class_result['Recall'] = 2 (normal_recall * 100)
        single_class_result['F1'] = 2 (normal_f1 * 100)
        single_class_result['Additive_Mode'] = 100 - additive_recall * 100
        single_class_result['Normal_Mode'] = 100 - normal_recall * 100

single_class_result = {'Normal': 0, 'Additive': 0}

normal_accuracy = 0.0
normal_correct = 0.0
for c in range(0, CATEGORIES):
    single_class_result = {'Normal': 0, 'Additive': 0}
    set_class_scores(c, single_class_result, normal_all_detected_class, additive_all_detected_class)
    normal_correct = normal_correct + normal_all_detected_class[c]
    normal_accuracy = normal_correct / test_data_categories[c]
    print('Class ', c, ' Normal Accuracy: ', normal_accuracy, ' %')
```

```
Class  0  Normal Accuracy:  0.0  %
Class  1  Normal Accuracy:  0.0  %
```

```
single_class_result = {'Normal': 0, 'Additive': 0}
normal_accuracy = 0.0
normal_correct = 0.0
for c in range(0, CATEGORIES):
    single_class_result = {'Normal': 0, 'Additive': 0}
    set_class_scores(c, single_class_result, normal_all_detected_class, additive_all_detected_class)
    normal_correct = normal_correct + normal_all_detected_class[c]
    normal_accuracy = normal_correct / test_data_categories[c]
    print('Class ', c, ' Normal Accuracy: ', normal_accuracy, ' %')
```

```
Class  0  Normal Accuracy:  0.0  %
Class  1  Normal Accuracy:  0.0  %
```

۷. جمع‌بندی

```
single_class_result = pd.DataFrame(single_class_result)
single_class_result.drop(columns=[0])
```

	Number	Accuracy	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Normal	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Additive	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

```
def calculate_macro(single_class_result):
    precision_sum = 0
    recall_sum = 0
    f1_sum = 0
    categories_sum = len(CATEGORIES)
    for c in CATEGORIES:
        if single_class_result['Precision'] < 1:
            recall_sum.append('Macro')
            continue
        precision = single_class_result['Precision'] + single_class_result[c, 'Precision']
        recall = single_class_result['Recall'] + single_class_result[c, 'Recall']
        f1 = single_class_result['F1'] + single_class_result[c, 'F1']
    all_class_result['Macro Precision'] = single_class_result['Precision'] / categories_sum
    all_class_result['Macro Recall'] = single_class_result['Recall'] / categories_sum
    all_class_result['Macro F1'] = single_class_result['F1'] / categories_sum

def calculate_macro(test_data, all_class_result):
    global all_class_result, normal_all_detected_class, normal_correct, test_data_categories
    normal_precision = normal_correct /normal_all_detected_class
    normal_recall = normal_correct / test_data_categories
    normal_f1 = 2 (normal_precision * normal_recall) / (normal_precision + normal_recall)
    additive_correct = additive_correct_detected_class / test_data_categories
    additive_all_detected = additive_correct_detected_class / test_data_categories
    additive_precision = additive_correct_detected_class / additive_all_detected
    additive_recall = additive_correct_detected_class / test_data_categories
    additive_f1 = 2 (additive_precision * additive_recall) / (additive_precision + additive_recall)
    all_class_result['Macro Precision'] = (normal_precision + additive_precision) / 2
    all_class_result['Macro Recall'] = (normal_recall + additive_recall) / 2
    all_class_result['Macro F1'] = (normal_f1 + additive_f1) / 2

calculate_macro(test_data, all_class_result)
print(all_class_result['Macro Precision'])
print(all_class_result['Macro Recall'])
print(all_class_result['Macro F1'])
```

```
0.0
0.0
0.0
```

```
def calculate_weighted(single_class_result, test_data, all_class_result):
    global all_class_result, normal_all_detected_class, normal_correct, test_data_categories
    weighted_precision = 0
    weighted_recall = 0
    weighted_f1 = 0
    for c in CATEGORIES:
        if single_class_result['Precision'] < 1:
            continue
        weighted_precision = weighted_precision + (test_data_categories[c] / sample) * single_class_result[c, 'Precision']
        weighted_recall = weighted_recall + (test_data_categories[c] / sample) * single_class_result[c, 'Recall']
        weighted_f1 = weighted_f1 + (test_data_categories[c] / sample) * single_class_result[c, 'F1']
    all_class_result['Weighted Precision'] = weighted_precision
    all_class_result['Weighted Recall'] = weighted_recall
    all_class_result['Weighted F1'] = weighted_f1

calculate_weighted(single_class_result, test_data, all_class_result)
```

```
calculate_macro(single_class_result, test_data, all_class_result)
calculate_weighted(single_class_result, test_data, all_class_result)
```

```
all_class_result = pd.DataFrame(all_class_result)
all_class_result
```

	Model	Macro Precision	Macro Recall	Macro F1	Weighted Precision	Weighted Recall	Weighted F1
Normal	Normal	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Additive	Additive	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

همانطور که مشاهده می‌کنید مقدار Accuracy برای هر دو مدل Additive و Normal برابر با صفر می‌باشد. دلیل این امر این است که مدل ما قادر به تشخیص هیچ داده‌ای نبوده و تمام داده‌ها را اشتباه تشخیص داده است.

۸. Additive Smoothing

```
train_df[train_df['categories'] == 1].head(20)[['categories', 'text_ids', 'Normal_Mode', 'Additive_Mode']]
```

	categories	text_ids	Normal_Mode	Additive_Mode
0	1	[illegible]	1.0	1.0
1	1	[illegible]	1.0	1.0
2	1	[illegible]	1.0	1.0
3	1	[illegible]	1.0	1.0
4	1	[illegible]	1.0	1.0
5	1	[illegible]	1.0	1.0
6	1	[illegible]	1.0	1.0
7	1	[illegible]	1.0	1.0
8	1	[illegible]	1.0	1.0
9	1	[illegible]	1.0	1.0
10	1	[illegible]	1.0	1.0
11	1	[illegible]	1.0	1.0
12	1	[illegible]	1.0	1.0
13	1	[illegible]	1.0	1.0
14	1	[illegible]	1.0	1.0
15	1	[illegible]	1.0	1.0
16	1	[illegible]	1.0	1.0
17	1	[illegible]	1.0	1.0
18	1	[illegible]	1.0	1.0
19	1	[illegible]	1.0	1.0

40000 rows × 3 columns

```
train_df[train_df['categories'] == 1].tail(20)[['categories', 'text_ids', 'Normal_Mode', 'Additive_Mode']]
```

	categories	text_ids	Normal_Mode	Additive_Mode
39980	1	[illegible]	1.0	1.0
39981	1	[illegible]	1.0	1.0
39982	1	[illegible]	1.0	1.0
39983	1	[illegible]	1.0	1.0
39984	1	[illegible]	1.0	1.0
39985	1	[illegible]	1.0	1.0
39986	1	[illegible]	1.0	1.0
39987	1	[illegible]	1.0	1.0
39988	1	[illegible]	1.0	1.0
39989	1	[illegible]	1.0	1.0
39990	1	[illegible]	1.0	1.0
39991	1	[illegible]	1.0	1.0
39992	1	[illegible]	1.0	1.0
39993	1	[illegible]	1.0	1.0
39994	1	[illegible]	1.0	1.0
39995	1	[illegible]	1.0	1.0
39996	1	[illegible]	1.0	1.0
39997	1	[illegible]	1.0	1.0
39998	1	[illegible]	1.0	1.0
39999	1	[illegible]	1.0	1.0

40000 rows × 3 columns